Project: Design XOR Gate

https://github.com/santhinagalla/Machine-Learning/tree/main/DeepLearning

- Step 1: Study the general idea on how to design XOR Gate
- Step 2: Using the following rules to design your own AND Gate, OR Gate, and NAND Gate.
    - - Forward process
        - Calculate the output Z for the given input (X,Y).
    - - Backward process
    - Adjust weights
        - If the output Z is too low, increase the weights by 0.5
        - which had inputs that were "1".
        - If the output Z is too high, decrease the weights by 0.5
        - which had inputs that were "1".
- Using step activation function
- Z := (W0 * C + W1 * X + W2 * Y >= T)

  where T := 1.0

  if (W0 * C + W1 * X + W2 * Y >= T)

    - then ouput is 1
    - else output = 0
- The bias C for NAND is 1.0

Step 2: Using the following rules to design your own AND Gate, OR Gate, and NAND Gate.

Train OR gate to get W1, W2, Y

Z := (W1 * X + W2 * Y >= T)

where T := 1.0.

Desired "OR" Function

x	y	z
0	0	0
0	1	1

1	0	1
1	1	1

##############################

Step 1 :- We might initially set the weights W1 and W2 to 0.0 and hope that by applying our training algorithm the weights will be trained to the appropriate values for the "OR" function.

Loop 1

W1=W2=0

Z := (0 * X + 0 * Y >= T)

Function

x	y	z
0	0	0
0	1	0
1	0	0
1	1	0

Step 2 :- As you can see, the weights were increased by 0.5 each time that the output for the input of (X,Y)=(1,1) gave an output Z less than the desired output Z of "1".

Loop 2

W1=W2=0.5

Z := (0.5 * X + 0.5 * Y >= T)

Function

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

Step 3 :- As you can see, the weights were increased by 0.5 each time that the output for the input of (X,Y)= (0,1) & (1,0) gave an output Z less than the desired output Z of "1".

Loop 3

W1=W2=1.0

Z := (1.0 * X + 1.0 * Y >= T)

Function

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Train AND gate to get W1, W2, Y

Z := (W1 * X + W2 * Y >= T)

where T := 1.0.

Desired "And" Function

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

##############################

Step 1 :- We might initially set the weights W1 and W2 to 0.0 and hope that by applying our training algorithm the weights will be trained to the appropriate values for the "And" function.

Loop 1

W1=W2=0

Z := (0 * X + 0 * Y >= T)

Function

x	y	z
0	0	0
0	1	0
1	0	0
1	1	0

Step 2 :- As you can see, the weights were increased by 0.5 each time that the output for the input of (X,Y)=(1,1) gave an output Z less than the desired output Z of "1". For (X,Y) of (0,0), (0,1), or (1,0), the output Z was always the desired value "0" so we did not need to increase or decrease the weights.

Loop 2

W1=W2=0.5

Z := (0.5 * X + 0.5 * Y >= T)

Function

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

==========

Train NAND gate to get W1, W2, Y

Z := (W0 * C + W1 * X + W2 * Y >= T)

where T := 1.0.

- The bias C for NAND is 1.0

Desired "NAND" Function

x	y	z
0	0	1
0	1	1

1	0	1
1	1	0

##

Step 1: Let's assume that our neural network already had a constant input C=1.0 but that its weight was fixed at 0.0 so we did not bother to show it previously since its weighted input would have been 1.0 * 0.0 = 0.0. We will now include C and allow W0 to be trained away from 0.0.

So, In Loop 1, W0 was set to 0.0 while W1 and W2 were at 0.5.

Loop 1

W0=0.0

W1=W2=0.5

Z := (0 * 1.0 + 0.5 * X + 0.5 * Y >= T)

Function

C	x	y	z
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Step 2 : - W0 was raised in Loop 1 when the input was (C,X,Y)=(1,0,0) and the output was Z=0 when we desired a higher output of Z=1. W1 and W2 were not raised for the same input because our algorithm specifies that we increase the weight when we want a higher output only when the input to that weight was a "1".

So, In Loop 2, As you can see, the weights were increased by 0.5 each time W0 was Increased by 0.5, while W1 and W2 were still at 0.5.

Loop 2

W0=0.5

W1=W2= 0.5

Z := (0.5 * 1.0 + 0.5 * X + 0.5 * Y >= T)

Function

C	x	y	z
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Step 3: - W0 was raised in Loop 2 when the input was (C, X, Y)=(1,0,0) and the output was Z=0 when we desired a higher output of Z=1. W1 and W2 were not raised for the same input because our algorithm specifies that we increase the weight when we want a higher output only when the input to that weight was a "1".

So, In Loop 3, As you can see, the weights were increased by 0.5 each time W0 was Increased by 1.0, while W1 and W2 were still at 0.5.

Loop 3

W0=1.0

W1=W2 = 0.5

Z := (1.0 * 1.0 + 0.5 * X + 0.5 * Y >= T)

Function

C	x	y	z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Step 4: - When the input (C, X, Y) was (1,1,1) for Loop 3, all three weights were decreased since the desired output of Z=0 was lower than the actual output of Z=1 and all three weights had a "1" as an input.

In Loop 4, W0 was remain same because, if we decrease W0 it effects the desired output for input (1,0,0). So, W1 and W2 decreased by 0.0.

Loop 4

W0=1.0

W1=W2 = 0.0

Z := (1.0 * 1.0 + 0.0 * X + 0.0 * Y >= T)

Function

C	x	y	z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Step 5: - When the input (C, X, Y) was (1,1,1) for Loop 4, all three weights were decreased since the desired output of Z=0 was lower than the actual output of Z=1 and all three weights had a "1" as an input.

In Loop 5, W0 was remain same because, if we decrease W0 it effects the desired output for input (1,0,0). So, W1 and W2 decreased by –0.5.

Loop 5

W0=1.0

W1=W2 = -0.5

Z := (1.0 * 1.0 + -0.5 * X + -0.5 * Y >= T)

Function

C	x	y	z
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Step 6: - when the input was (C, X, Y)=(1,0,1) & (1,1,0) for Loop 5 and the output was Z=0 when we desired a higher output of Z=1. W0, W1 and W2 were raised for the same input because our algorithm specifies that we increase the

weight when we want a higher output only when the input to that weight was a "1".

In Loop 6, W0 was increased by 1.5. As W1 & W2 were not changed because, if we Increase W1 & W2 it effects the desired output for input (1,1,1). So, W1 and W2 are same.

Loop 6

W0=1.5

W1=W2 = -0.5

Z := (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= T)

Function

C	x	y	z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 3: Please answer

- What is the formula for
  - Z1 := X "AND" Y
  - **Z1 := (0.5 * X + 0.5 * Y >= 1.0)**
- What is the formula for
  - Z1 := X "OR" Y
  - **Z1 := (1.0 * X + 1.0 * Y >= 1.0)**
- What is the formula for
  - Note:

    Bias is +1.5

    C = 1; W0 = 1.5; W1=W2 = -0.5
  - Z2 := X "NAND" Y

- **Z2 := (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)**
- **Z2 := (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)**
- **Z2 := (- 0.5 * Y >= 0.5 * X + -1.5 * 1.0 + 1.0)**
- **Z2 := (-0.5 * Y >= 0.5 * X – 0.5)**
- **Z2 := (Y <= -X + 1.0)**

- What is the formula for
  - Z1 := X "Or" Y
  - Z2 := X "NAnd" Y
  - Z := Z3 := Z1 "AND" Z2
  - **Z := (X "Or" Y) "And" (X "NAnd" Y)**
  - **Z := (1.0 * X + 1.0 * Y >= 1.0) "And"**

    (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)
  - **Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) +**

    0.5 * (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Step 4: Please prove that your designed XOR Gate work

Our neural network equation can be created by combining neural equations.

Z1 := X "Or" Y

Z2 := X "NAND" Y

Z := Z3 := Z1 "AND" Z2

Z := (X "Or" Y) "AND" (X "NAND" Y)

Z := (1.0 * X + 1.0 * Y >= 1.0) "And"

(1.5 + -0.5 * X + -0.5 * Y >= 1.0)

Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) +

0.5 * (1.5 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Please prove that the neural network works for XOR Gate

A.X=1, Y=1

Z := (0.5 * (1.0 * 1.0 + 1.0 * 1.0 >= 1.0) +

0.5 * (1.5 + -0.5 * 1.0 + -0.5 * 1.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 + 1.0 >= 1.0) + 0.5 * (1.5 + -0.5 + -0.5 >= 1.0) >= 1.0)

Z := (0.5 * (2.0 >= 1.0) + 0.5 * (0.5 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (false) >= 1.0)

Z := (0.5 * 1 + 0.5 * 0 >= 1.0)

Z := (0.5 + 0.0 >= 1.0)

Z := (false)

Z := 0

B. X=1, Y=0

Z := (0.5 * (1.0 * 1.0 + 1.0 * 0.0 >= 1.0) +

0.5 * (1.5 + -0.5 * 1.0 + -0.5 * 0.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 + 0.0 >= 1.0) + 0.5 * (1.5 + -0.5 + -0.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 >= 1.0) + 0.5 * (1.0 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (true) >= 1.0)

Z := (0.5 * 1 + 0.5 * 1 >= 1.0)

Z := (0.5 + 0.5 >= 1.0)

Z := (true)

Z := 1

C. X=0, Y=1

Z := (0.5 * (1.0 * 0.0 + 1.0 * 1.0 >= 1.0) +

0.5 * (1.5 + -0.5 * 0.0 + -0.5 * 1.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 + 1.0 >= 1.0) + 0.5 * (1.5 + -0.0 + -0.5 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 >= 1.0) + 0.5 * (1.0 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (true) >= 1.0)

Z := (0.5 * 1 + 0.5 * 1 >= 1.0)

Z := (0.5 + 0.5 >= 1.0)

Z := (true)

Z := 1

D .X=0, Y=0

Z := (0.5 * (1.0 * 0.0 + 1.0 * 0.0 >= 1.0) +

0.5 * (1.5 + -0.5 * 0.0 + -0.5 * 0.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 + 0.0 >= 1.0) + 0.5 * (1.5 + -0.0 + -0.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 >= 1.0) + 0.5 * (1.5 >= 1.0) >= 1.0)

Z := (0.5 * (false) + 0.5 * (true) >= 1.0)

Z := (0.5 * 0 + 0.5 * 1 >= 1.0)

Z := (0.0 + 0.5 >= 1.0)

Z := (false)

Z := 0

Hence, proved that the neural network equation can be created by combining neural equations.

OR		NAND		XOR
X Y | Z1		X Y | Z2		X Y | Z3
---		---		---
0 0 | 0		0 0 | 1		0 0 | 0
0 1 | 1	AND	0 1 | 1	=	0 1 | 1
1 0 | 1		1 0 | 1		1 0 | 1
1 1 | 1		1 1 | 0		1 1 | 0